

```
1  # Importing required libraries
2  from sqlalchemy import create_engine, Column, Float, Integer, String
3  from sqlalchemy.orm import sessionmaker, declarative_base
4  import sqlite3
5  import pandas as pd
6  import numpy as np
7  import matplotlib.pyplot as plt
8  import os
9  import pathlib
10 import math
11 import warnings
12 import unittest
13 from bokeh.plotting import figure, show
14 from bokeh.io import output_notebook
15 from bokeh.models import ColumnDataSource
16 import subprocess
17 import os
18 '''
19 SECTION 1.2
20 '''
21 # Suppressing warnings
22 warnings.filterwarnings("ignore", category=UserWarning)
23
24 # Defining the base class for SQLAlchemy
25 Base = declarative_base()
26
27 # Defining the TrainingData class
28 class TrainingData(Base):
29     __tablename__ = 'train_data'
30     id = Column(Integer, primary_key=True)
31     x = Column(Float)
32     y1 = Column(Float)
33     y2 = Column(Float)
34     y3 = Column(Float)
35     y4 = Column(Float)
36
37 # Defining the DataVisualization class
38 class DataVisualization:
39     @staticmethod
40     def plot_train_data_scatter():
41         """
42         Plot scatter plot of training data.
43
44         This function connects to the SQLite database, retrieves training data,
45         and plots a scatter plot
46         for visualizing the relationship between 'X' and 'Y1', 'Y2', 'Y3',
47         'Y4'.
```

```
48     None
49
50     Returns:
51     None
52     """
53     try:
54         # Connecting to SQLite database
55         conn = sqlite3.connect('DataBase.db')
56         query = 'SELECT * FROM train_data'
57         df = pd.read_sql_query(query, conn)
58         x = df['x']
59         y1 = df['y1']
60         y2 = df['y2']
61         y3 = df['y3']
62         y4 = df['y4']
63
64         # Plotting scatter plot
65         plt.title('\n "X" against Y1, Y2, Y3, Y4 from trained data \n',
66 fontdict={'fontsize': 20, 'fontweight': 5, 'color': 'Black'})
67         plt.xlabel("X values", fontdict={'fontsize': 10, 'fontweight': 10,
68 'color': 'blue'})
69         plt.ylabel("Y Values", fontdict={'fontsize': 10, 'fontweight': 10,
70 'color': 'blue'})
71
72         plt.scatter(x, y1, alpha=1, s=5, c='blue', label='X vs Y1')
73         plt.scatter(x, y2, alpha=1, s=5, c='red', label='X vs Y2')
74         plt.scatter(x, y3, alpha=1, s=5, c='green', label='X vs Y3')
75         plt.scatter(x, y4, alpha=1, s=5, c='yellow', label='X vs Y4')
76
77         plt.legend()
78         plt.show()
79
80         # Closing database connection
81         conn.close()
82     except sqlite3.Error as e:
83         print(f"SQLite error: {e}")
84     except Exception as e:
85         print(f"An error occurred: {e}")
86
87 # Defining the DataProcessing class
88 class DataProcessing:
89     def find_sum_of_least_squares(self, x, y):
90         """
91         Find the sum of least squares between two arrays.
92
93         Parameters:
94         x (numpy.array): First array.
95         y (numpy.array): Second array.
96
97         Returns:
98         float: Sum of least squares.
```

```
96         """
97         try:
98             x = np.array(x)
99             y = np.array(y)
100             difference = np.subtract(x, y)
101             square = np.square(difference)
102             sum_of_squares = np.sum(square)
103             return sum_of_squares
104         except Exception as e:
105             print(f"An error occurred: {e}")
106
107     def find_least_squares(self, x, y):
108         """
109         Find the least squares between two arrays.
110
111         Parameters:
112         x (numpy.array): First array.
113         y (numpy.array): Second array.
114
115         Returns:
116         pandas.DataFrame: DataFrame containing least squares values.
117         """
118         try:
119             x = np.array(x)
120             y = np.array(y)
121             difference = np.subtract(x, y)
122             square = np.square(difference)
123             return pd.DataFrame(square)
124         except Exception as e:
125             print(f"An error occurred: {e}")
126
127     def find_deviation(self, x, y):
128         """
129         Find the deviation between two arrays.
130
131         Parameters:
132         x (numpy.array): First array.
133         y (numpy.array): Second array.
134
135         Returns:
136         pandas.DataFrame: DataFrame containing deviation values.
137         """
138         try:
139             x = np.array(x)
140             y = np.array(y)
141             difference = np.subtract(x, y)
142             return pd.DataFrame(difference)
143         except Exception as e:
144             print(f"An error occurred: {e}")
145
146     def any_deviation_greater_than_threshold(self, x, y, threshold):
```

```

147         """
148         Check if any deviation is greater than the threshold.
149
150         Parameters:
151         x (numpy.array): First array.
152         y (numpy.array): Second array.
153         threshold (float): Threshold value.
154
155         Returns:
156         bool: True if any deviation is greater than the threshold, False
otherwise.
157         """
158         try:
159             x = np.array(x)
160             y = np.array(y)
161             difference = pd.DataFrame(np.subtract(x, y))
162             return (difference > threshold).any().any()
163         except Exception as e:
164             print(f"An error occurred: {e}")
165
166 # Defining the DataLoader class
167 class DataLoader:
168     def __init__(self, data_base, table_name, data_frame):
169         self.Base = Base
170         self.data_base = data_base
171         self.table_name = table_name
172         self.data_frame = data_frame
173         self.engine = create_engine('sqlite:///{}.db'.format(self.data_base))
174
175     def load_data(self):
176         """
177         Load data into the SQLite database.
178
179         Parameters:
180         None
181
182         Returns:
183         None
184         """
185         try:
186             self.Base.metadata.create_all(self.engine)
187             table_name = self.table_name
188             self.data_frame.to_sql(table_name, con=self.engine,
if_exists='replace', index=False)
189         except Exception as e:
190             print(f"An error occurred: {e}")
191
192     def close_connection(self):
193         """
194         Close the SQLite database connection.
195

```

```
196         Parameters:
197         None
198
199         Returns:
200         None
201         """
202         try:
203             self.engine.dispose()
204         except Exception as e:
205             print(f"An error occurred: {e}")
206
207 # Defining the Ideal class
208 class Ideal(DataProcessing):
209     def load_ideal_data(self):
210         """
211         Load ideal data from the CSV file.
212
213         Parameters:
214         None
215
216         Returns:
217         pandas.DataFrame: Loaded ideal data.
218         """
219         try:
220             self.ideal = pd.read_csv("Datasets/ideal.csv")
221             return self.ideal
222         except FileNotFoundError as e:
223             print(f"File not found: {e}")
224         except Exception as e:
225             print(f"An error occurred: {e}")
226
227     def find_ideal(self, x):
228         """
229         Find the ideal functions based on least squares.
230
231         Parameters:
232         x (numpy.array): Array for which ideal functions are calculated.
233
234         Returns:
235         pandas.DataFrame: DataFrame containing ideal functions.
236         """
237         try:
238             database = DataProcessing()
239             least_squares = [database.find_sum_of_least_squares(x,
self.ideal.iloc[:, i]) for i in range(len(self.ideal.columns))]
240             first_four_least_squares = sorted(least_squares)[:4]
241             indices = [least_squares.index(i) for i in
first_four_least_squares]
242             ideal_functions = [self.ideal.iloc[:, i] for i in indices]
243             ideal_functions = pd.DataFrame(ideal_functions).transpose()
244             ideal_functions.columns = ['y1', 'y2', 'y3', 'y4']
```

```
245         return ideal_functions
246     except Exception as e:
247         print(f"An error occurred: {e}")
248
249 # Defining the Test class
250 class Test(DataProcessing):
251     def load_test_data(self):
252         """
253         Load test data from the CSV file.
254
255         Parameters:
256         None
257
258         Returns:
259         pandas.DataFrame: Loaded test data.
260         """
261         try:
262             self.test = pd.read_csv("Datasets/test.csv")
263             return self.test
264         except FileNotFoundError as e:
265             print(f"File not found: {e}")
266         except Exception as e:
267             print(f"An error occurred: {e}")
268
269 # Defining the Train class
270 class Train(DataProcessing):
271     def load_training_data(self):
272         """
273         Load training data from the CSV file.
274
275         Parameters:
276         None
277
278         Returns:
279         pandas.DataFrame: Loaded training data.
280         """
281         try:
282             self.train = pd.read_csv("Datasets/train.csv")
283             return self.train
284         except FileNotFoundError as e:
285             print(f"File not found: {e}")
286         except Exception as e:
287             print(f"An error occurred: {e}")
288
289     def get_deviation(self, x):
290         """
291         Get deviation between x and y values in the training data.
292
293         Parameters:
294         x (numpy.array): Array for which deviation is calculated.
295
```

```
296         Returns:
297         pandas.DataFrame: DataFrame containing deviation values.
298         """
299         try:
300             self.deviation = self.find_deviation(x, self.train.iloc[:, 1])
301             return self.deviation
302         except Exception as e:
303             print(f"An error occurred: {e}")
304
305     # Defining the TestYourCode class
306     class TestYourCode(unittest.TestCase):
307         def test_find_sum_of_least_squares(self):
308             data_processing = DataProcessing()
309             x = [1, 2, 3, 4, 5]
310             y = [2, 4, 6, 8, 10]
311             result = data_processing.find_sum_of_least_squares(x, y)
312             self.assertEqual(result, 55)
313
314         def test_find_least_squares(self):
315             data_processing = DataProcessing()
316             x = [1, 2, 3, 4, 5]
317             y = [2, 4, 6, 8, 10]
318             result = data_processing.find_least_squares(x, y)
319             expected_result = pd.DataFrame([1, 4, 9, 16, 25], columns=['x'])
320             pd.testing.assert_frame_equal(result, expected_result)
321
322         def test_find_deviation(self):
323             data_processing = DataProcessing()
324             x = [1, 2, 3, 4, 5]
325             y = [2, 4, 6, 8, 10]
326             result = data_processing.find_deviation(x, y)
327             expected_result = pd.DataFrame([-1, -2, -3, -4, -5], columns=['x'])
328             pd.testing.assert_frame_equal(result, expected_result)
329
330         def test_any_deviation_greater_than_threshold(self):
331             data_processing = DataProcessing()
332             x = [1, 2, 3, 4, 5]
333             y = [2, 4, 6, 8, 10]
334             threshold = 5
335             result = data_processing.any_deviation_greater_than_threshold(x, y,
threshold)
336             self.assertTrue(result)
337
338     # Main block of code
339     if __name__ == "__main__":
340         try:
341             # Defining file paths and names
342             path = pathlib.Path(__file__).parent.resolve()
343             dataset_path = os.path.join(path, "Datasets")
344             ideal_filename = "ideal"
345             train_filename = "train"
```

```
346     database_name = "DataBase"
347
348     # Reading and loading ideal data
349     df_ideal = pd.read_csv(os.path.join(dataset_path, "{0}.csv".format(ideal_filename)))
350     ideal_table_name = ideal_filename + "_data"
351     ideal_data_loader = DataLoader(database_name, ideal_table_name,
df_ideal)
352     ideal_data_loader.load_data()
353     ideal_data_loader.close_connection()
354
355     # Reading and loading training data
356     df_train = pd.read_csv(os.path.join(dataset_path, "{0}.csv".format(train_filename)))
357     train_table_name = train_filename + "_data"
358     train_data_loader = DataLoader(database_name, train_table_name,
df_train)
359     train_data_loader.load_data()
360     train_data_loader.close_connection()
361
362     # Creating instances of classes
363     train = Train()
364     test = Test()
365     ideal = Ideal()
366     data_processing = DataProcessing()
367
368     # Loading data
369     train_data = train.load_training_data()
370     test_data = test.load_test_data()
371     ideal_data = ideal.load_ideal_data()
372
373     # Finding ideal functions
374     ideal_functions = ideal.find_ideal(train_data.iloc[:, 1])
375     ideal_functions.insert(0, 'x', train_data.iloc[:, 0])
376
377     # Calculating deviations
378     deviation_between_training_and_ideal = pd.DataFrame([])
379     for column in ideal_functions.columns:
380         deviation_between_training_and_ideal[column] =
data_processing.find_deviation(train_data.iloc[:, 1], ideal_functions[column])
381     deviation_between_training_and_ideal =
pd.DataFrame(deviation_between_training_and_ideal)
382
383     # Calculating absolute deviation
384     absolute_deviation = deviation_between_training_and_ideal.abs()
385     maximum_deviation = absolute_deviation.max().max()
386
387     # Calculating sqrt(2) * maximum deviation
388     sqrt_2 = math.sqrt(2)
389     sqrt_2_maximum_deviation = sqrt_2 * maximum_deviation
390
```



```
391         # Initializing arrays for x and y values
392         x_values = np.array([])
393         y1_values = np.array([])
394         y2_values = np.array([])
395         y3_values = np.array([])
396         y4_values = np.array([])
397
398         # Finding best fit values
399         for t in test_data.iloc[:, 0]:
400             least_squares = np.array([(x - t) ** 2 for x in
ideal_functions.iloc[:, 0]])
401
402             index = np.argmin(least_squares)
403             x_values = np.append(x_values, ideal_functions.iloc[index, 0])
404             y1_values = np.append(y1_values, ideal_functions.iloc[index, 1])
405             y2_values = np.append(y2_values, ideal_functions.iloc[index, 2])
406             y3_values = np.append(y3_values, ideal_functions.iloc[index, 3])
407             y4_values = np.append(y4_values, ideal_functions.iloc[index, 4])
408
409         # Creating DataFrame for best fit values
410         best_fit_values = pd.DataFrame([x_values, y1_values, y2_values,
y3_values, y4_values]).transpose()
411         best_fit_values.columns = ['x', 'y1', 'y2', 'y3', 'y4']
412
413         # Creating DataVisualization object
414         data_visualization = DataVisualization()
415
416         # Initializing lists for DataFrames
417         y1 = pd.DataFrame([])
418         y2 = pd.DataFrame([])
419         y3 = pd.DataFrame([])
420         y4 = pd.DataFrame([])
421
422         table_list = [y1, y2, y3, y4]
423
424         # Creating DataFrames for each ideal function
425         for i in range(1, 5):
426             if data_processing.any_deviation_greater_than_threshold(
427                 best_fit_values.iloc[:, i], test_data.iloc[:, 1],
sqrt_2_maximum_deviation):
428                 print('y' + str(i) + ' is not in range')
429             else:
430                 table_list[i - 1]['x'] = test_data.iloc[:, 0]
431                 table_list[i - 1]['y'] = test_data.iloc[:, 1]
432                 table_list[i - 1]['delta'] =
data_processing.find_deviation(test_data.iloc[:, 1],
433
best_fit_values.iloc[:, i])
434                 table_list[i - 1]['ideal function'] = best_fit_values.iloc[:,
i]
435
```

```
436         # Loading test data into SQLite database
437         test_loader = DataLoader(database_name, 'test_' + 'y' + str(i +
1), table_list[i - 1])
438         test_loader.load_data()
439
440         # Loading training data and ideal functions into SQLite database
441         training_loader = DataLoader(database_name, 'training_data',
best_fit_values)
442         training_loader.load_data()
443         training_loader.close_connection()
444
445         ideal_loader = DataLoader(database_name, 'ideal_functions', ideal_data)
446         ideal_loader.load_data()
447         ideal_loader.close_connection()
448
449         # Plotting the scatter plot
450         data_visualization.plot_train_data_scatter()
451
452     except Exception as e:
453         print(f"An error occurred: {e}")
454
455
456
457     '''
458     SECTION 1.3
459     '''
460     # Set your GitHub username and repository name
461     github_username = "Name22" #please enter your github username
462     repository_name = "test"#please enter your repo name in github
463     repository_url = f"https://github.com/{github_username}/{repository_name}.git"
464
465     # Set the directory where you want to initialize the repository
466     repository_directory = "C:\\Users\\Name\\Downloads\\Python Assignment-2\\Python
Assignment\\Python Assignment\\Task 2 Assignment" #Refer to this to amend the
directory on your local machine
467
468     # Change to the repository directory
469     os.chdir(repository_directory)
470
471     # Initialize a new Git repository
472     subprocess.run(["git", "init"])
473
474     # Add all files to the repository
475     subprocess.run(["git", "add", "."])
476
477     # Commit the changes
478     subprocess.run(["git", "commit", "-m", "Initial commit"])
479
480     # Add the GitHub remote repository
481     subprocess.run(["git", "remote", "add", "origin", repository_url])
482
```

```
483 # Push the changes to the 'develop' branch
484 result_push = subprocess.run(["git", "push", "-u", "origin", "develop"])
485
486 # Print the result of pushing changes
487 print("Result of pushing changes:", result_push)
488
489 # Print a message indicating success
490 print("Changes pushed successfully.")
491
492
```